AAiT

Computer Architecture & Organization

Chapter 9

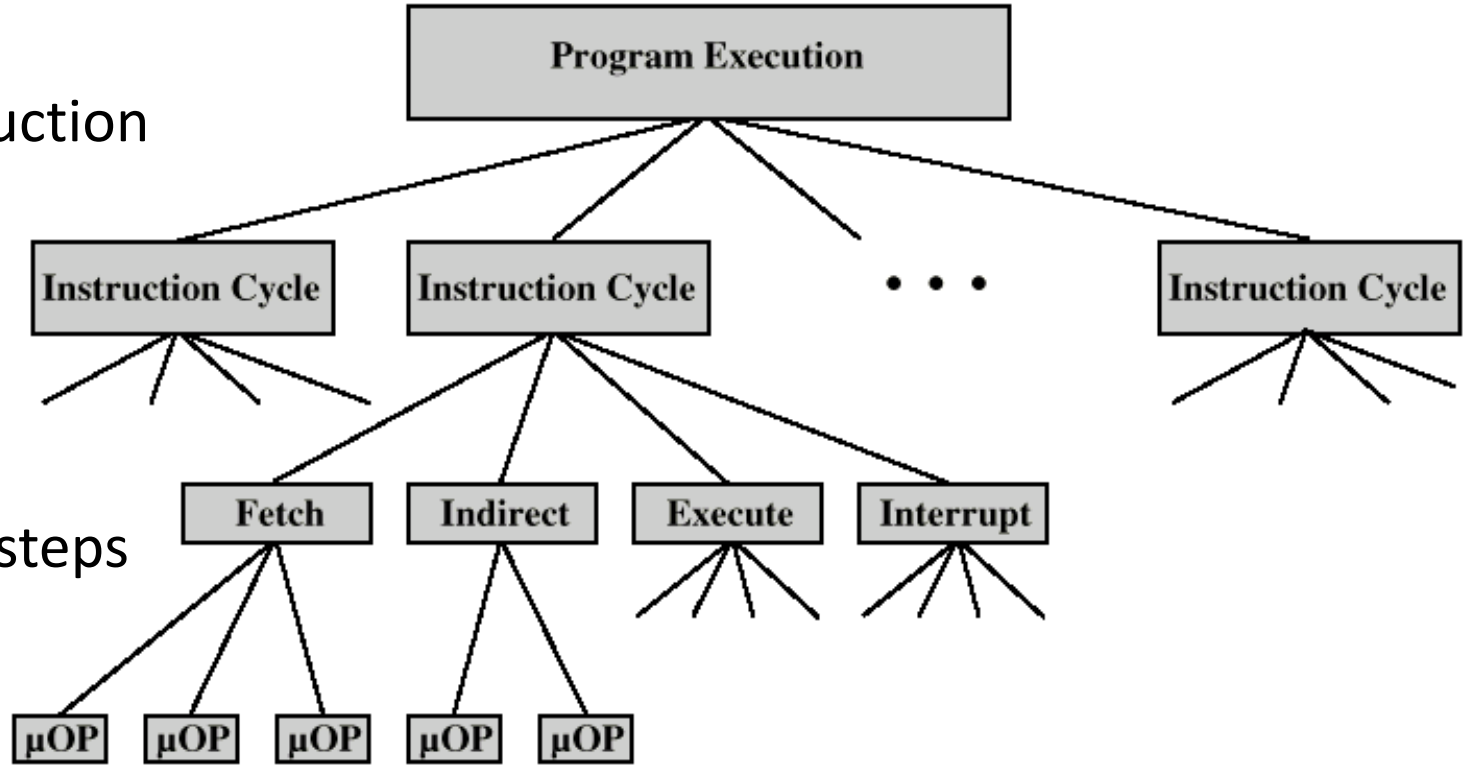
Control Unit Operation

Functional requirements for a processor

- The following list are things needed to specify the function of a processor
  1. Operations (opcodes)
  2. Addressing modes
  3. Registers
  4. I/O module interface
  5. Memory module interface
  6. Interrupts
- Items 1 through 3 are defined by the instruction set.
- Items 4 and 5 are typically defined by specifying the system bus.
- Item 6 is defined partially by the system bus and partially by the type of support the processor offers to the operating system
- In this chapter we see how the various elements of the processor are controlled to provide these functions.

Micro-Operations

- A computer executes a program
- Program execution involve Instruction cycles: Fetch, indirect, execute interrupt … Sub cycles



- Each Sub cycle has a number of steps (see pipelining) called **micro-operations**
- **micro-operations** are atomic operation of CPU

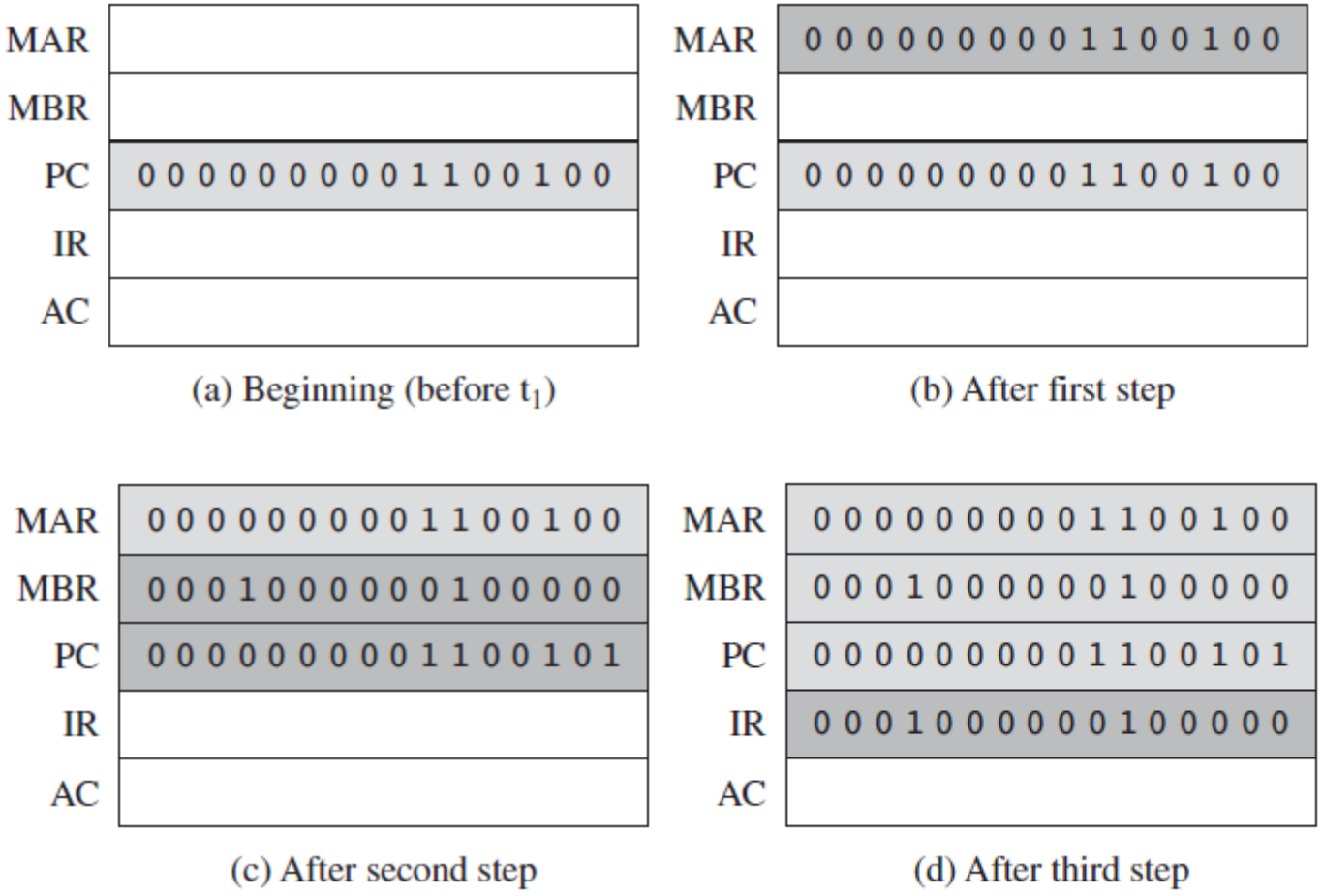
Fetch - 4 Registers

- Memory Address Register (MAR)
  - Connected to address bus
  - Specifies address for read or write op
- Memory Buffer Register (MBR)
  - Connected to data bus
  - Holds data to write or last data read
- Program Counter (PC)
  - Holds address of next instruction to be fetched
- Instruction Register (IR)
  - Holds last instruction fetched

Fetch Sequence

- Address of next instruction is in PC move it to MAR.
  - Address (MAR) is placed on address bus automatically.
- Control unit issues READ command
  - Result (data from memory) appears on data bus
  - Data from data bus copied into MBR
- PC incremented by 1 (in parallel with data fetch from memory)
- Data (instruction) moved from MBR to IR
  - MBR is now free for further data fetches

Fetch Sequence (Example)



t_1: MAR ← (PC)
t_2: MBR ← Memory
PC ← (PC) + I
t_3: IR ← (MBR)

OR

t_1: MAR ← (PC)
t_2: MBR ← Memory
t_3: PC ← (PC) + I
IR ← (MBR)

Fetch cycle actually consists of three steps and four microoperations.

Rules for Clock Cycle Grouping

- Proper sequence must be followed
  - MAR <- (PC) must precede MBR <- (memory)
- Conflicts must be avoided
  - Must not read & write same register at same time
  - MBR <- (memory) & IR <- (MBR) must not be in same cycle
- Also: PC <- (PC) +1 involves addition
  - ALU can be used
  - May need additional micro-operations

Indirect Cycle

MAR <- ($IR_{address}$) - address field of IR
MBR <- (memory)
$IR_{address}$ <- ($MBR_{address}$)

- MBR contains an address
- IR is now in same state as if direct addressing had been used
- (What does this say about IR size?)

Interrupt Cycle

t1:MBR <-(PC)

t2:MAR <- save-address

PC <- routine-address

t3:memory <- (MBR)

- This is a minimum
  - May be additional micro-ops to get addresses
  - N.B. saving context is done by interrupt handler routine, not micro-ops

Execute Cycle (ADD)

- Different for each instruction
- Example

  ADD R1,X - add the contents of location X to Register R1 , result in R1

  t1: MAR <- ($IR_{address}$)

  t2: MBR <- (memory)

  t3: R1 <- R1 + (MBR)

- Note no overlap of micro-operations

Example Execute Cycle (ISZ)

- ISZ X - increment and skip if zero

  t1: MAR <- ($IR_{address}$)

  t2: MBR <- (memory)

  t3: MBR <- (MBR) + 1

  t4: memory <- (MBR)

  if (MBR) == 0 then PC <- (PC) + 1

- Notes:
  - if is a single micro-operation
  - The test and skip micro-operation is done during t4

Example Execute Cycle (BSA)

- BSA X - Branch and save address
  - Address of instruction following BSA is saved in X
  - Execution continues from X+1

t1: MAR <- ($IR_{address}$)
MBR <- (PC)

t2: PC <- ($IR_{address}$)
memory <- (MBR)

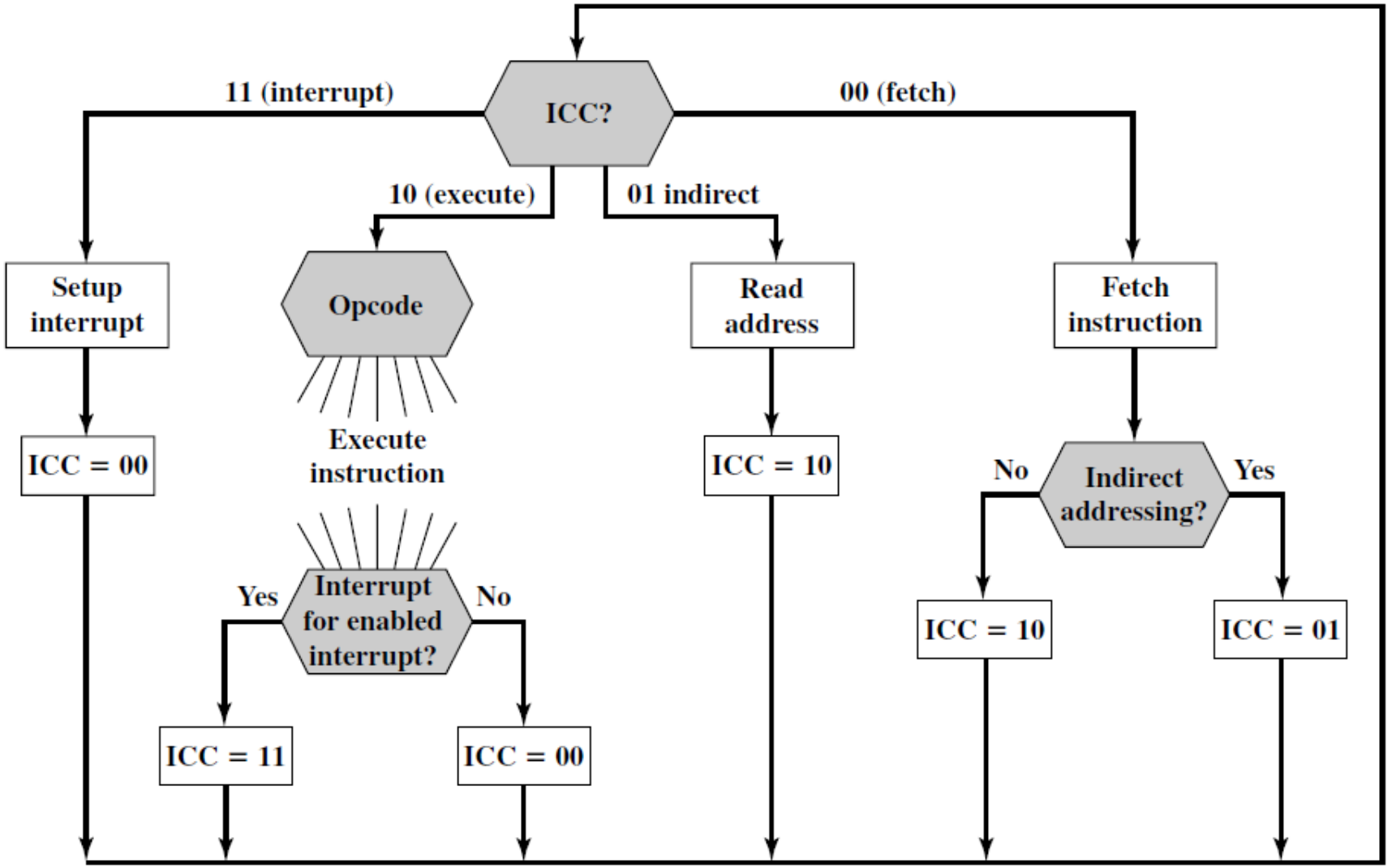
t3: PC <- (PC) + 1

Instruction Cycle

- Each phase decomposed into sequence of elementary micro-operations

  E.g. fetch, indirect, and interrupt cycles
- Execute cycle
  - One sequence of micro-operations for each opcode
- Need to tie sequences together
- Assume new 2-bit register
  - Instruction cycle code (ICC) designates which part of cycle processor is in
    - 00: Fetch
    - 01: Indirect
    - 10: Execute
    - 11: Interrupt

Flowchart for Instruction Cycle



Three-step process to characterize the control unit

- **Define basic elements of processor**
  - ALU
  - Registers
  - Internal data paths
  - External data paths
  - Control unit
- **Describe micro-operations processor performs**
  - Transfer data between registers
  - Transfer data from register to external
  - Transfer data from external to register
  - Perform arithmetic or logical ops
- **Determine functions control unit must perform**

Sequencing

Causing the CPU to step through a series of micro-operations

Execution

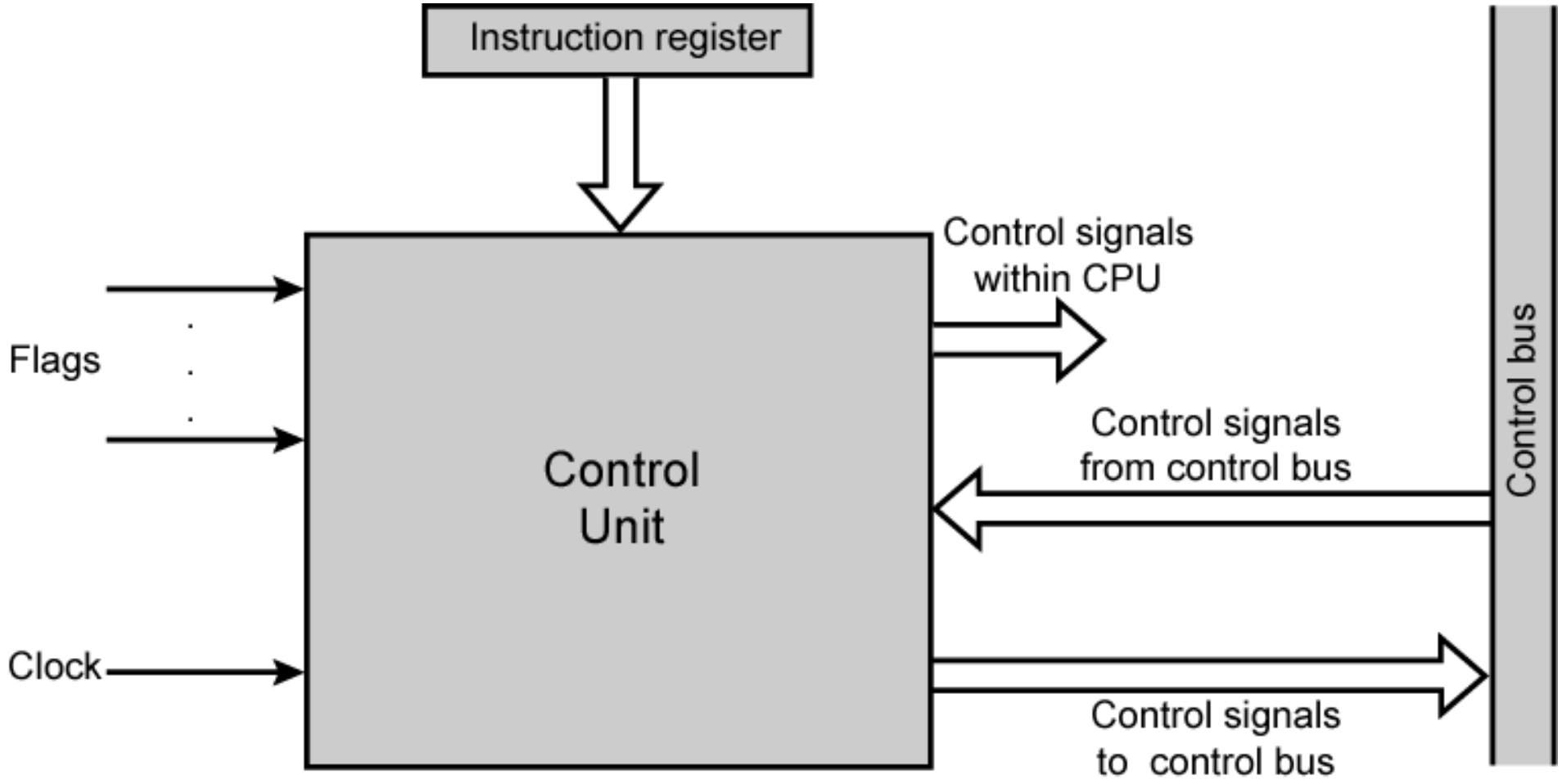
Causing the performance of each micro-op

This is done using Control Signals

Control Signals

- Clock
  - One micro-instruction (or set of parallel micro-instructions) per clock cycle
- Instruction register
  - Op-code for current instruction
  - Determines which micro-instructions are performed
- Flags
  - State of CPU
  - Results of previous operations
- From control bus
  - Interrupts
  - Acknowledgements

Model of Control Unit



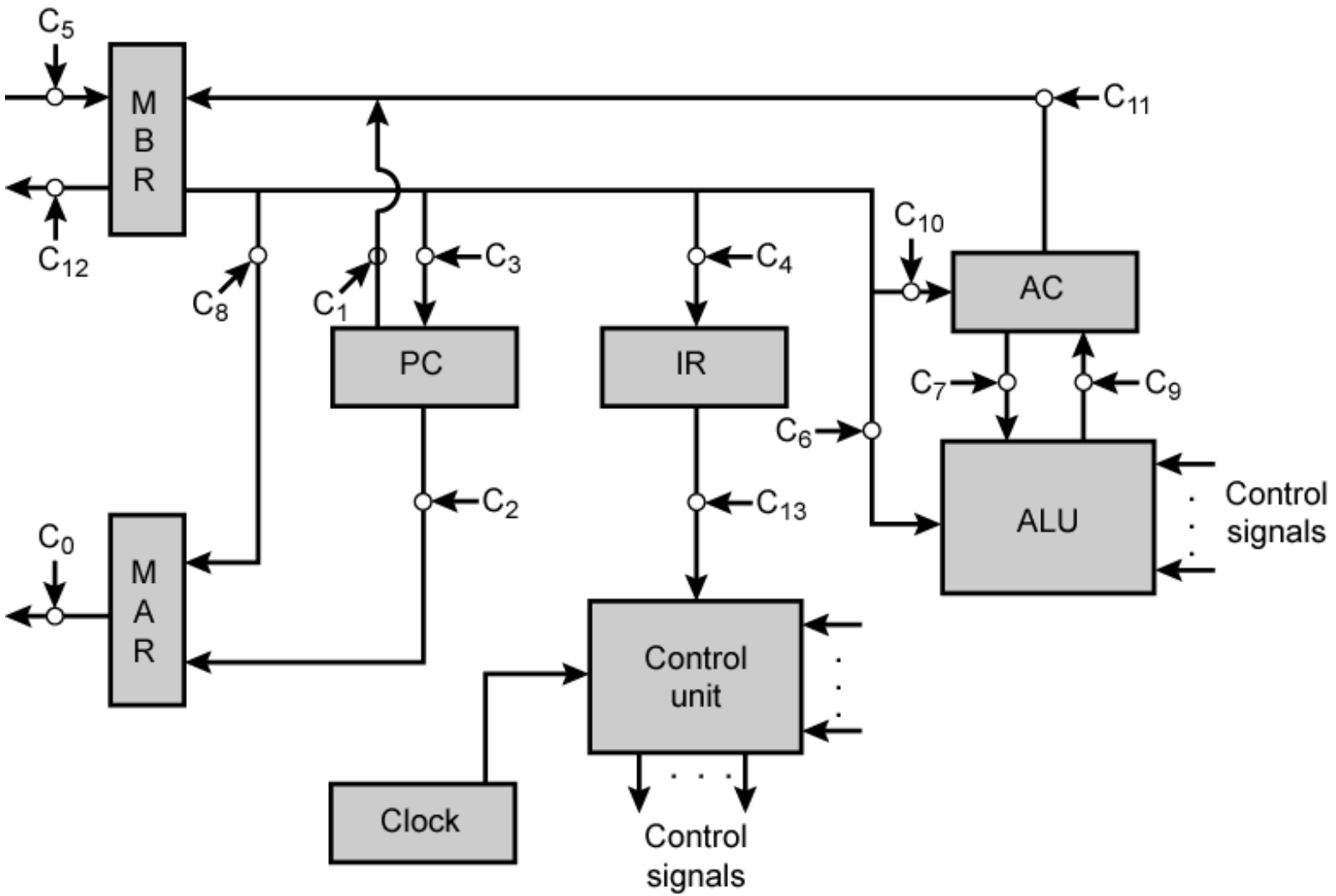
Control Signals - output

- Within CPU
  - Cause data movement
  - Activate specific functions
- Via control bus
  - To memory
  - To I/O modules

Example Control Signal Sequence - Fetch

- MAR <- (PC)
  - Control unit activates signal to open gates between PC and MAR
- MBR <- (memory)
  - Open gates between MAR and address bus
  - Memory read control signal
  - Open gates between data bus and MBR

Data Paths and Control Signals



Control signals go to three separate destinations:

Data paths
ALU
System bus

Internal Organization

- Usually a single internal bus
- Gates control movement of data onto and off the bus
- Control signals control data transfer to and from external systems bus
- Temporary registers needed for proper operation of ALU

AAiT

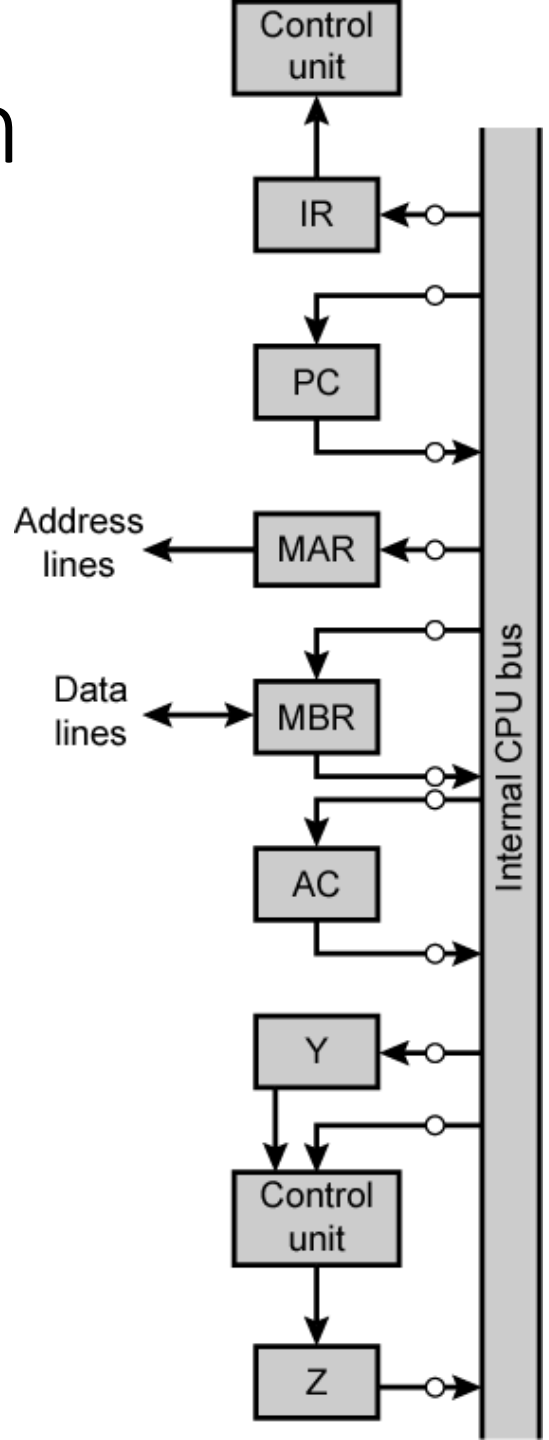
Micro-operations and Control Signals

	Micro-operations	Active Control Signals
Fetch:	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Memory PC ← (PC) + 1	C_5, C_R
	t_3: IR ← (MBR)	C_4
Indirect:	t_1: MAR ← (IR(Address))	C_8
	t_2: MBR ← Memory	C_5, C_R
	t_3: IR(Address) ← (MBR(Address))	C_4
Interrupt:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Save-address PC ← Routine-address	
	t_3: Memory ← (MBR)	C_{12}, C_W

C_R = Read control signal to system bus.

C_W = Write control signal to system bus.

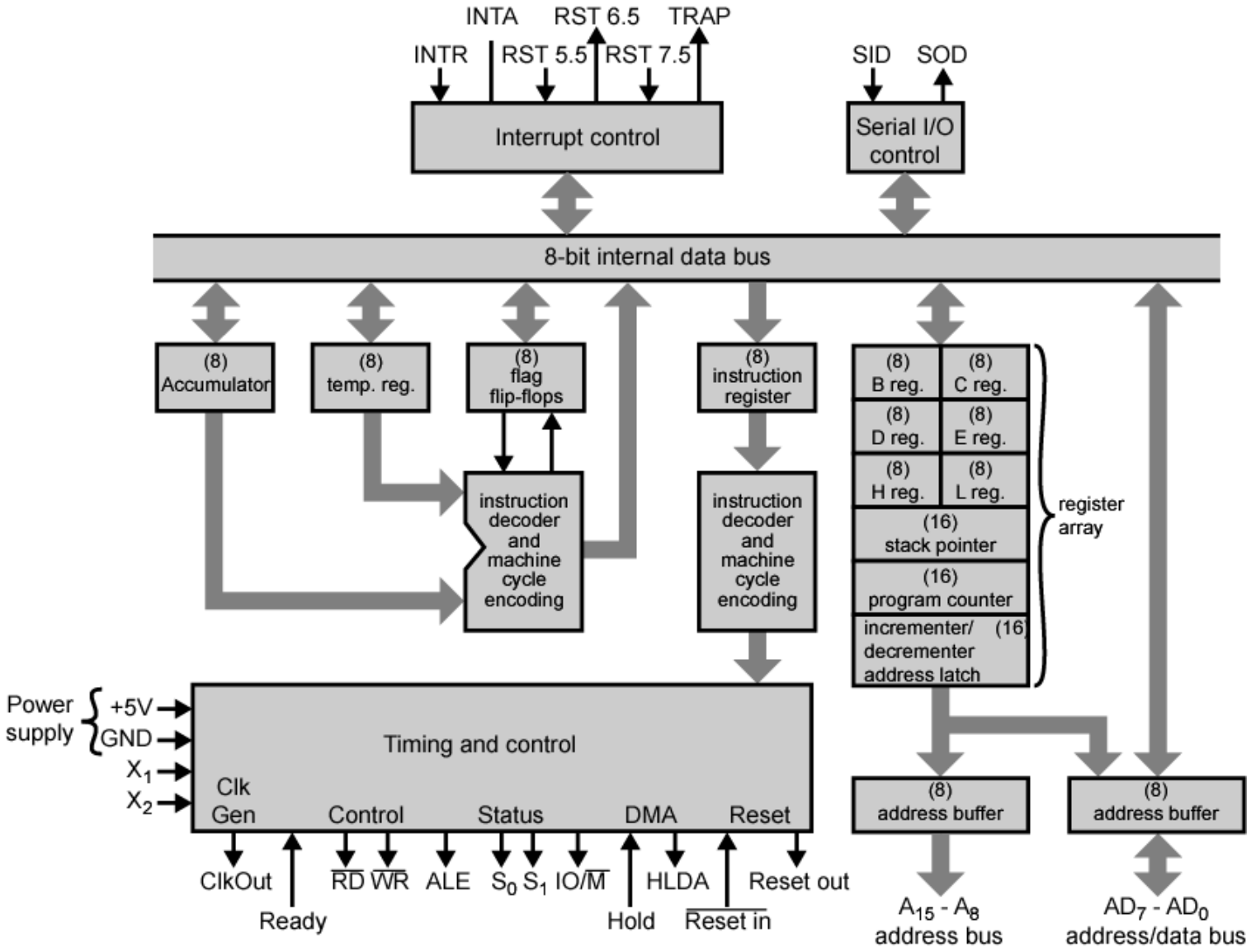
CPU with Internal Bus



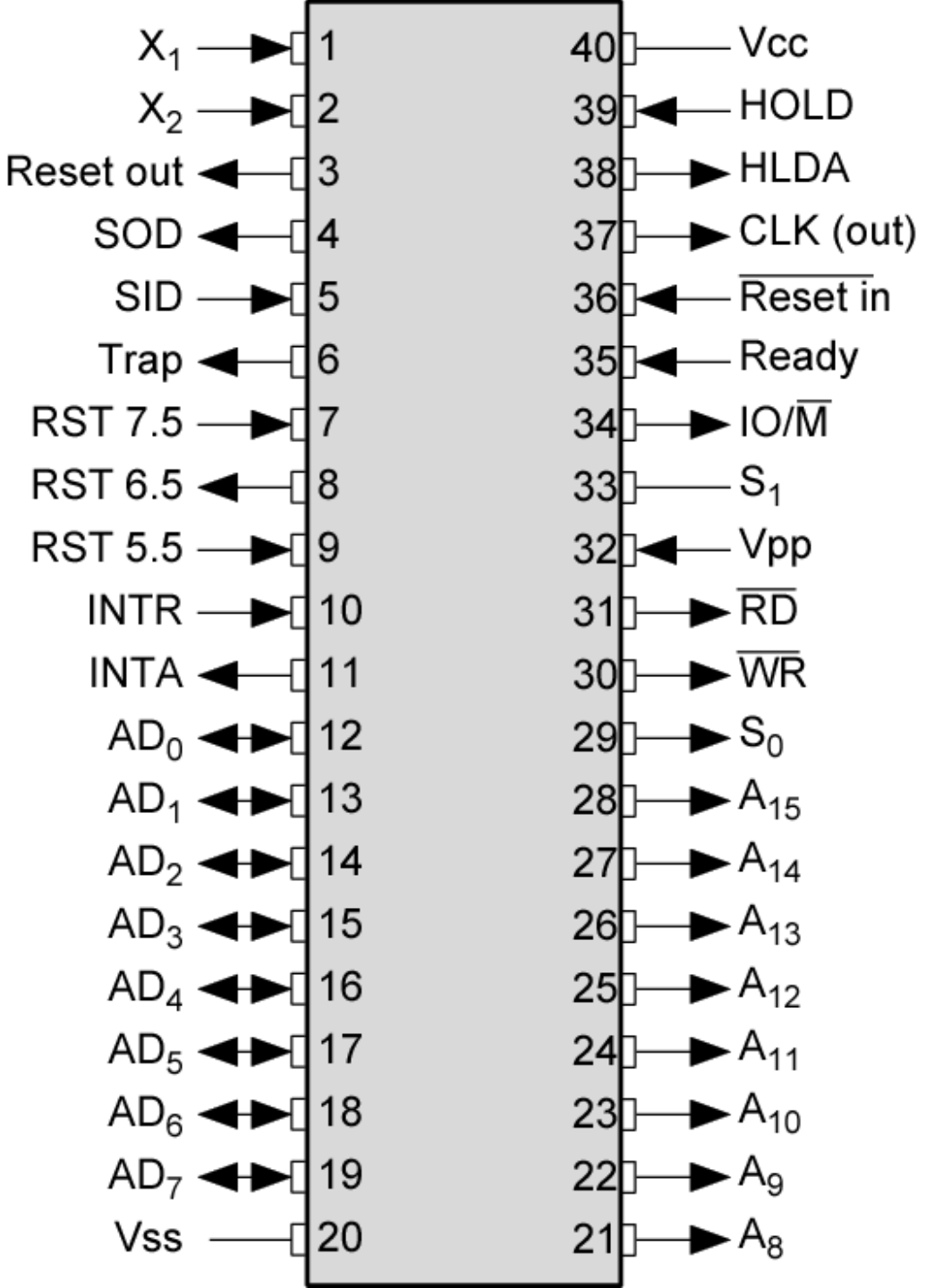
- Two new registers, labeled Y and Z, have been added to the organization.
- The ALU is a combinatorial circuit
- Output of the ALU cannot be directly connected to the bus, because this output would feed back to the input; register Z provides temporary output storage
- With this arrangement, an operation to add a value from memory to the AC would have the following steps:

t_1: MAR ← (IR(address))
t_2: MBR ← Memory
t_3: Y ← (MBR)
t_4: Z ← (AC) + (Y)
t_5: AC ← (Z)

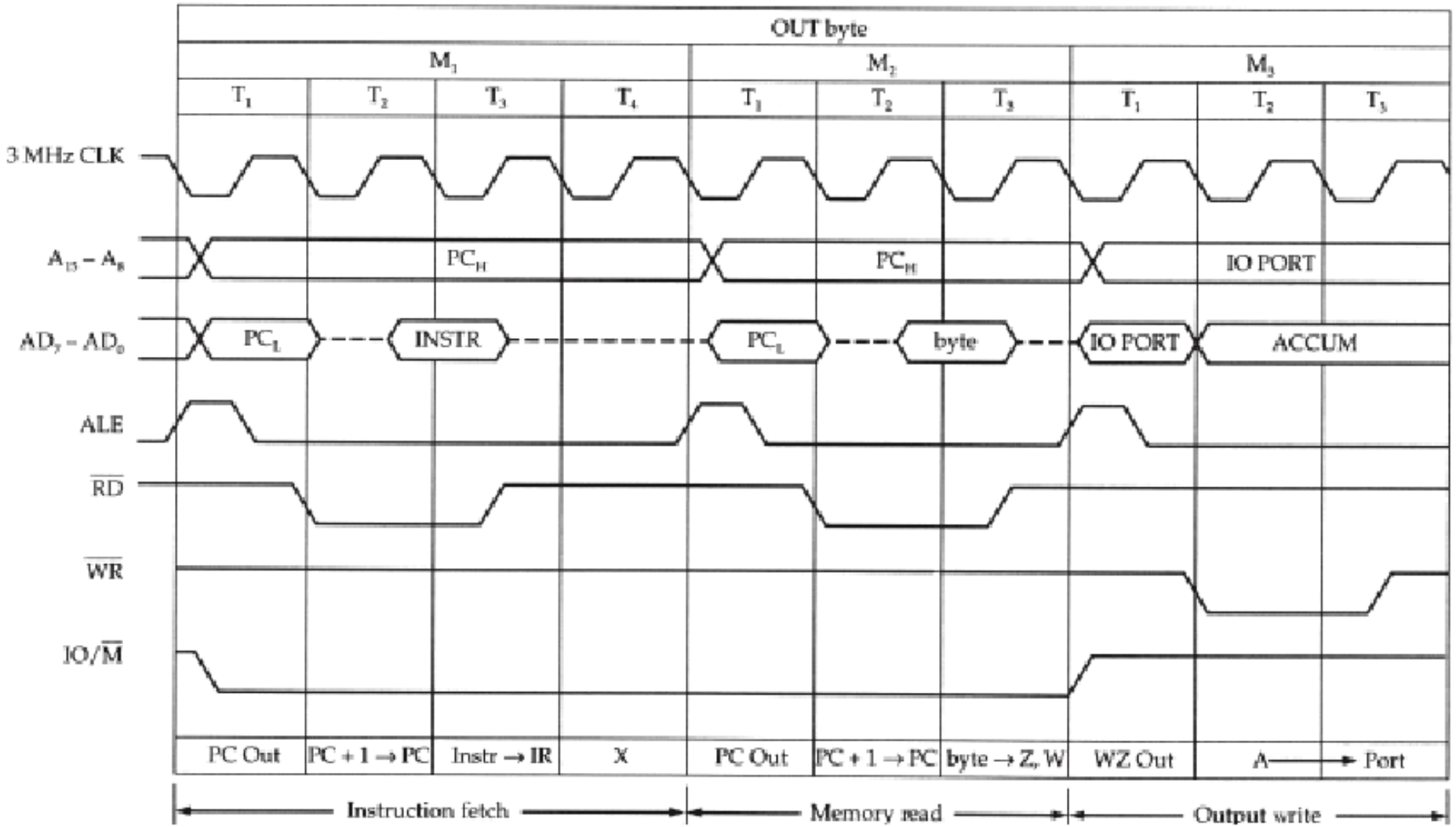
Intel 8085 CPU Block Diagram



Intel 8085 Pin Configuration



Intel 8085 OUT Instruction Timing Diagram



Hardwired Implementation

The Control Unit can be implemented in two ways:

1. Hardwired control
2. Microprogrammed control (chapter 10)

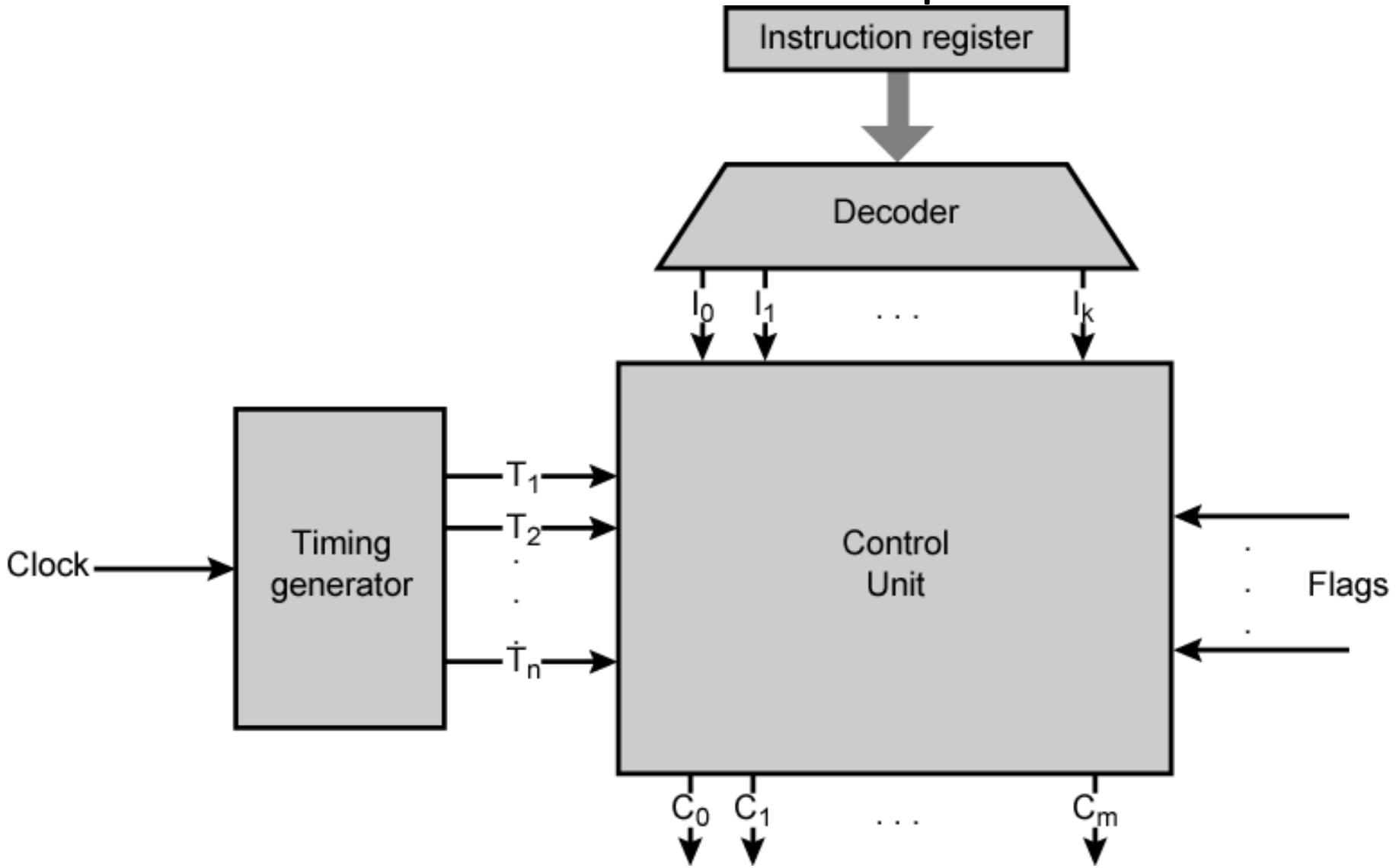
Control unit inputs

- Flags and control bus
  - Each bit means something
- Instruction register
  - Op-code causes different control signals for each different instruction
  - Unique logic for each op-code
  - Decoder takes encoded input and produces single output
  - *n* binary inputs and 2^n outputs

Hardwired Implementation

- Clock
  - Repetitive sequence of pulses
  - Useful for measuring duration of micro-ops
  - Must be long enough to allow signal propagation
  - Different control signals at different times within instruction cycle
  - Need a counter with different control signals for t1, t2 etc.

Control Unit with Decoded Inputs



Problems With Hard Wired Designs

- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Difficult to add new instructions